|  |
| --- |
| MongoDB connections using mongoose with nodejs + expressjs |
| //mongodb connections  mongoose.connect(url)  const con = mongoose.connection  con.on('open', () => {  console.log('crud database connected...')  })  Remove everywhere {  useNewUrlParser: true,  useUnifiedTopology: true,  }); |
| ****Using**** async/await ****for Cleaner Code**** This approach ensures cleaner and more modern JavaScript syntax.  const mongoose = require('mongoose');  const connectDB = async () => {  try {  await mongoose.connect('mongodb://localhost:27017/crud', {  useNewUrlParser: true,  useUnifiedTopology: true,  });  console.log('CRUD database connected...');  } catch (error) {  console.error('Database connection failed:', error);  process.exit(1); // Exit process with failure  }  };  connectDB(); |
| 2. ****Using**** .then ****and**** .catch ****Promises**** This uses promise chaining for error handling.  const mongoose = require('mongoose');  mongoose.connect('mongodb://localhost:27017/crud', {  useNewUrlParser: true,  useUnifiedTopology: true,  }).then(() => {  console.log('CRUD database connected...');  }).catch((error) => {  console.error('Database connection failed:', error);  }); |
| 3. ****Encapsulating Connection Logic in a Separate Module**** This modular approach helps in reusability.  **db.js**  const mongoose = require('mongoose');  const connectDB = () => {  return mongoose.connect('mongodb://localhost:27017/crud', {  useNewUrlParser: true,  useUnifiedTopology: true,  });  };  module.exports = connectDB;  app.js  const connectDB = require('./db');  connectDB()  .then(() => console.log('CRUD database connected...'))  .catch((error) => console.error('Database connection failed:', error)); |
| 4. ****Using Mongoose Events for Connection Monitoring**** This listens for connection events like error or disconnected.  const mongoose = require('mongoose');  mongoose.connect('mongodb://localhost:27017/crud', {  useNewUrlParser: true,  useUnifiedTopology: true,  });  const db = mongoose.connection;  db.on('connected', () => {  console.log('CRUD database connected...');  });  db.on('error', (error) => {  console.error('Connection error:', error);  });  db.on('disconnected', () => {  console.log('Database disconnected');  }); |
| 5. ****Using Environment Variables for Configuration**** This makes the connection URL dynamic and secure.  .env  MONGO\_URI=mongodb://localhost:27017/crud  app.js  require('dotenv').config();  const mongoose = require('mongoose');  mongoose.connect(process.env.MONGO\_URI, {  useNewUrlParser: true,  useUnifiedTopology: true,  });  const db = mongoose.connection;  db.once('open', () => {  console.log('CRUD database connected...');  });  db.on('error', (error) => {  console.error('Connection error:', error);  }); |
| Key Notes:  * **Error Handling**: Always handle connection errors to avoid application crashes. * **Options**: useNewUrlParser and useUnifiedTopology ensure compatibility with modern MongoDB drivers. * **Environment Variables**: Using .env for storing sensitive information like connection strings is a best practice. |
| ****Other Event Listeners for Mongoose Connection****  | **Event** | **Description** | | --- | --- | | connected | Triggered when the connection is successfully established. | | error | Triggered when a connection error occurs. | | disconnected | Triggered when the connection to the database is lost. | | reconnected | Triggered when Mongoose reconnects to the database after losing connection. | | close | Triggered when the connection is closed manually or by the MongoDB server. | | open | Triggered when the connection is ready for use. | | fullsetup | Triggered when all nodes of a replica set are connected. | | all | Triggered when all nodes in a sharded cluster are connected. | |
| ****Best Practices****  1. **Error Handling**: Always listen to the error event to handle connection issues gracefully. 2. **Clean Shutdown**: On disconnected, consider implementing reconnection logic or proper cleanup. 3. **Use Promises**: Prefer mongoose.connect() with async/await or .then for better readability and error management. 4. **Event Debugging**: Use db.on('all', callback) to log all events during development. |
| ****1. Add**** .env ****to**** .gitignore  * Open or create a .gitignore file in the root of your project directory. * Add the following line to it:  .env****2. Verify**** .gitignore ****is in Effect**** If the .env file has already been committed to your repository, you need to untrack it. Follow these steps: a. Remove the .env file from the Git index:git rm --cached .env This will remove the .env file from the Git tracking while keeping it on your local filesystem. b. Commit the change:git commit -m "Stop tracking .env file"****3. Push the Changes**** Push the updated .gitignore to your remote repository: git push****4. Double-Check**** After these steps, Git will no longer track changes to the .env file, and it will remain local to your development environment. Make sure .env is excluded from future commits. |